Master's Degree in Informatics Engineering
HIGH PERFORMANCE COMPUTING
LERIDA MONSO, JOSEP LLUIS
MPI IMPLEMENTATION
KIN TAT, TAN

MPI IMPLEMENTATION

KIN TAT, TAN

Master's Degree in Informatics Engineering
HIGH PERFORMANCE COMPUTING
LERIDA MONSO, JOSEP LLUIS
MPI IMPLEMENTATION
KIN TAT, TAN

Universitat de Lleida

Sumário

Master's Degree in Informatics Engineering
HIGH PERFORMANCE COMPUTING
LERIDA MONSO, JOSEP LLUIS
MPI IMPLEMENTATION
KIN TAT, TAN

Universitat de Lleida

GitHub

https://github.com/tankintat/highperformanceudl

Development the Central Idea

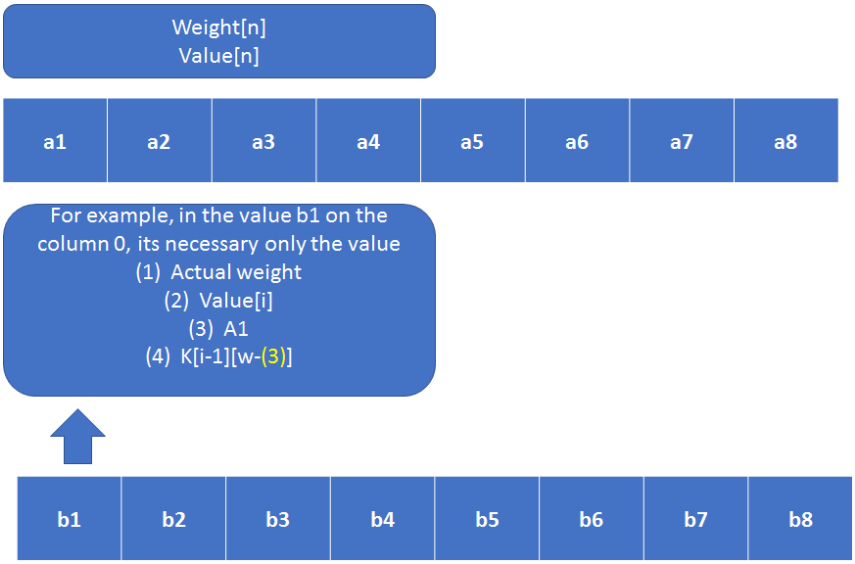
The problem resume basically these align for:

```
for (i = 0; i <= N; i++){
        for (w = 0; w <= W; w++) {
                if (i==0 || w==0)
                        K[i][w] = 0;
                else if (wt[i-1] <= w)
                                K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);
                else
                        K[i][w] = K[i-1][w];
        }
}
```

Table 1- Serial Knapsack problem.

The idea about this problem its use sequentially information about the previous state to trying to find the best way generating the next state, and go on. But if we realize we just use some variables about the previous state, (1) the actual weight, (2) the actual value, (3) the value In the previous state on the same column, (4) and the value which the difference between actual w minus the value (3).

So, because this we don't need to send all previous line to generate the next values to fill the array. Figure to represent:



Master's Degree in Informatics Engineering
HIGH PERFORMANCE COMPUTING
LERIDA MONSO, JOSEP LLUIS
MPI IMPLEMENTATION
KIN TAT, TAN

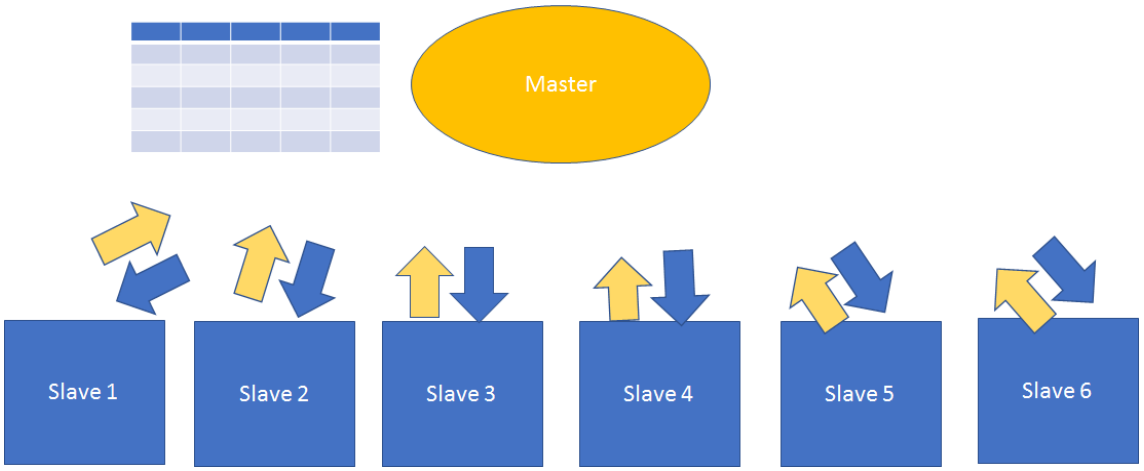
Universitat de Lleida

Mpi description solution

The solution involves concentrate the array values on the master cluster and the slaves treat only process the result, and let all the services to the master cluster.

This reduces the amount of data size that will be transferred from a clustered cluster.

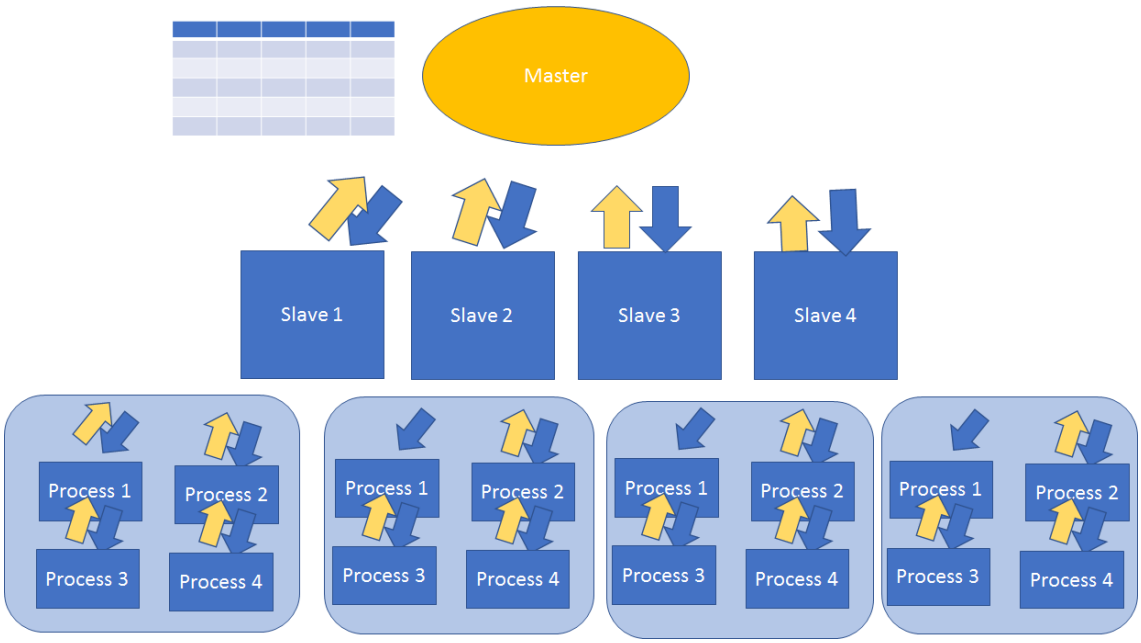
So, the master will send the array to each slave and then returns the result value back, will persistence in the master cluster.



Master's Degree in Informatics Engineering
HIGH PERFORMANCE COMPUTING
LERIDA MONSO, JOSEP LLUIS
MPI IMPLEMENTATION
KIN TAT, TAN

Universitat de Lleida

The idea to make the hybrid its generate for example 4 clusters with the MPI and each cluster have to control process OpenMP to returns to the master cluster. That way we can expected better performance.



Development

On the master just prepare the array to send to the slaves, and wait the new result and put in the array.

```
if (rank==0){
    printf("Valor de Width e NItems: %i %i\n", Width, Nitems);
    for (i = 1; i <= Nitems; i++){
        for (w = 1; w <= Width; w++){

            message[0] = wt[i-1];
            message[1] = w;
            message[2] = val[i-1];
            if(w>=wt[i-1])
                message[3] = K[i-1][w-wt[i-1]];
            else
                message[3] = 0;
            message[4] = K[i-1][w];
            message[5] = i;

            cluster = (cluster % quantitycluster) + 1;
            printf("%i Cluster: %i, Mensagem enviada: %i, %i, %i, %i, %i\n", i, cluster, message[0], message[1], message[2], message[3], message[4]);
            rc = MPI_Isend(message, 10, MPI_INT, cluster, 1, MPI_COMM_WORLD, &req);
            MPI_WAIT(&req, &status);
            rc = MPI_Irecv(returned, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &req);
            MPI_WAIT(&req, &status);
            rc = MPI_Get_count(&status, MPI_DOUBLE, &count);

            K[i][w] = returned[0];
            fflush(stdout);
            /*MPI_WAIT(&req, &status);*/
        }
        MPI_WAIT(&req, &status);
    }
```

And then use one for to send to the slaves a flag to stop hearing signals.

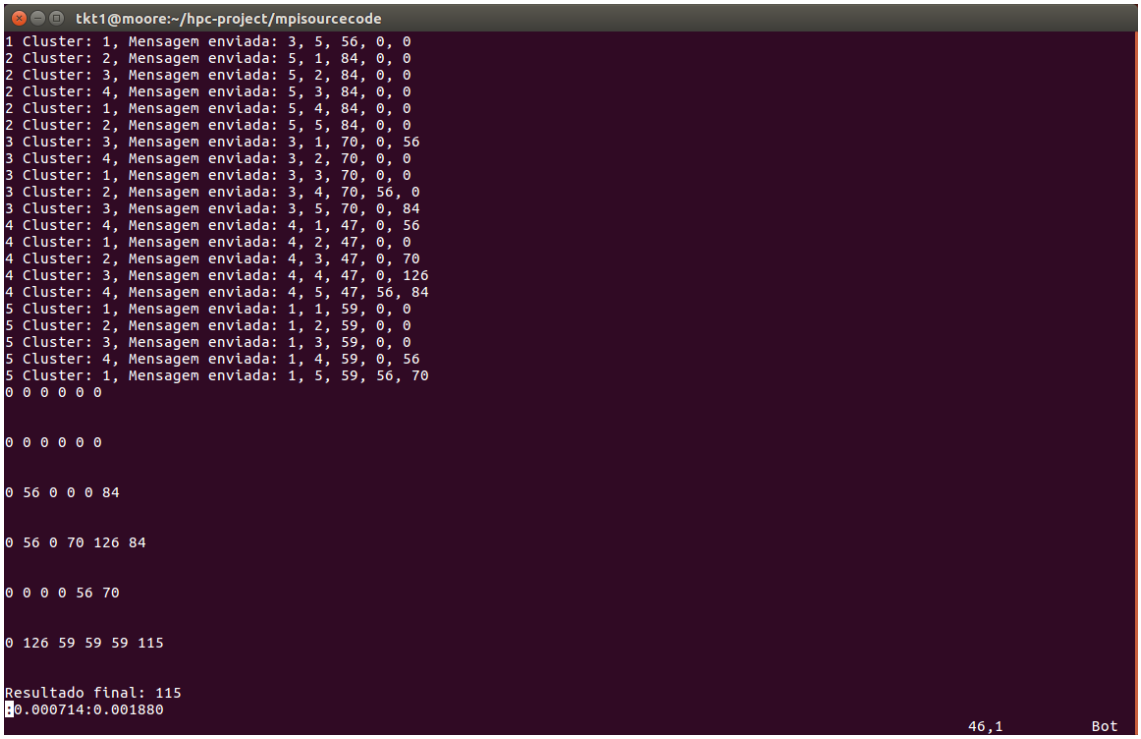
```
message[0] = -1;
for(i=1;i<=quantitycluster;i++)
    rc = MPI_Send(message, 10, MPI_INT, i, 1, MPI_COMM_WORLD);
```

On the slaves will receive the data, and process the data and returns to the master.

```
else {
    MPI_Status status;

    rc = MPI_Irecv(message, 10, MPI_INT, 0, MPI_ANY_TAG, MPI_COMM_WORLD, &req);
    while(message[0] != -1){
        if (message[0] <= message[1]){
            if((message[2] + message[3]) && (message[2] + message[3])  >= message[4])
                returned[0] = (message[2] + message[3]);
            else
                returned[0] = message[4];
        }
        else
            returned[0] = message[4];

        /*K[message[5]][message[1]] = returned[0];*/

        fflush(stdout);
        rc = MPI_Isend(returned, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, &req);
        MPI_WAIT(&req, &status);

        rc = MPI_Get_count(&status, MPI_DOUBLE, &count);
        rc = MPI_Irecv(message, 10, MPI_INT, 0, MPI_ANY_TAG, MPI_COMM_WORLD, &req);
        MPI_WAIT(&req, &status);
    }
}
```

The idea was using the MPI_Isend and MPI_Irecv to generate the slaves assincron and then use the MPI_Wait to wait return on the slaves.

But don't returns in the theory.



In the first time, the final result was good cause a luck reason the clusters finish synchrony.

Conclusions

Of course, working with MPI needs a greater concern in managing the threads, I lost a lot of time in executing a process and did not return anything, but the reason was because of the number of submits that were running. What was lacking in this work was to identify the barrier for the completion of previous processes before starting the next ones.